Familienname:

Vorname:

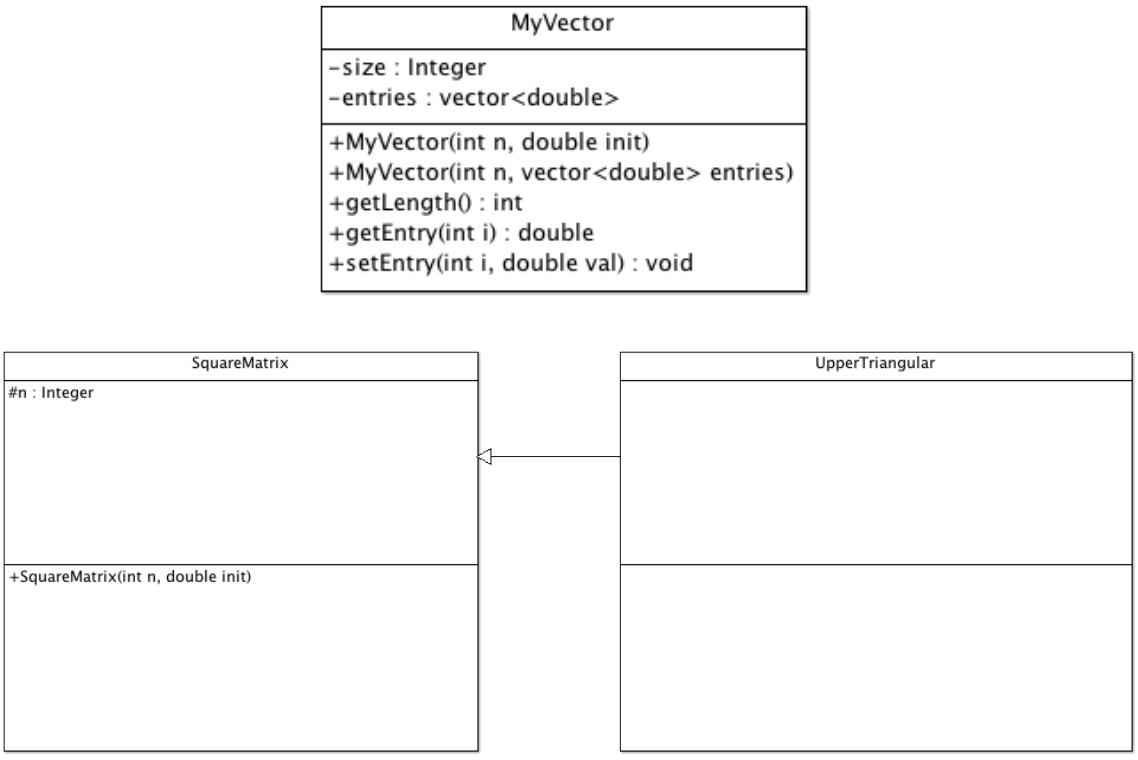
Matrikelnummer:

Aufgabe 1 (4 Punkte):
Aufgabe 2 (1 Punkte):
Aufgabe 3 (1 Punkte):
Aufgabe 4 (3 Punkte):
Aufgabe 5 (4 Punkte):
Aufgabe 6 (5 Punkte):
Aufgabe 7 (2 Punkte):
Aufgabe 8 (2 Punkte):
Aufgabe 9 (2 Punkte):
Aufgabe 10 (6 Punkte):

Gesamtpunktzahl:

Schriftlicher Test zu C++ (90 Minuten)
VU Einführung ins Programmieren für TM

22. Juni 2012



Oben sehen Sie UML Diagramme für die Klassen, die im Laufe der folgenden Aufgaben verwendet werden sollen. Die Klasse `MyVector` kann (ohne dass Sie diese implementieren sollen) mit ihrer vollen Funktionalität verwendet werden, d.h. Sie dürfen auf alle Methoden zurückgreifen die Sie obigem Diagramm entnehmen können. Die beiden übrigen Klassen `SquareMatrix` und `UpperTriangular` sind Großteils in den folgenden Aufgaben zu implementieren, bzw. die UML Diagramme auszufüllen.

Aufgabe 1 (4 Punkte). Schreiben Sie die Klassendefinition für eine Klasse `SquareMatrix` zur Speicherung quadratischer Matrizen $A \in \mathbb{R}^{n \times n}$. Diese soll die Dimension $n > 0$ explizit speichern. Die Koeffizienten der Matrix sollen außerdem als `vector<double>`, d.h. einfach indiziert und spaltenweise gespeichert werden. Eine 3×3 Matrix

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

werde also beispielsweise in einem Vektor `entries` mit

$$\texttt{entries} = (a_{00}, a_{10}, a_{20}, a_{01}, a_{11}, a_{21}, a_{02}, a_{12}, a_{22})$$

gespeichert. Die Klasse soll außerdem über folgende Methoden verfügen:

- ein Konstruktor, der eine $n \times n$ Matrix mit einem gegebenen Initialwert initialisiert
- ein Konstruktor, dem neben der Dimension n auch ein `vector<double>` mit Einträgen übergeben wird und der damit eine entsprechende Matrix initialisiert
- eine Methode `getN`, welche die Dimension ausgibt
- eine Methode `getEntry`, welche zu gegebenen Indizes den entsprechenden Eintrag zurückliefert
- eine Methode `setEntry`, welche an den gegebenen Indizes einen übergebenen Wert einträgt
- eine Methode, mit deren Hilfe eine Matrix-Vektor-Multiplikation mit einem Vektor $b \in \mathbb{R}^n$ vom Typ `MyVector` mittels $x = A * b$ ausgeführt werden kann. Der Ergebnisvektor $x \in \mathbb{R}^n$ soll wieder vom Typ `MyVector` sein

Füllen Sie außerdem das obenstehende UML Diagramm zur Klasse `SquareMatrix` analog zum UML Diagramm von `MyVector` aus, d.h. tragen Sie alle Felder und Methoden mit voller Signatur und entsprechenden Zugriffsspezifizierern ein.
Hinweis: An dieser Stelle ist **nur** die Klassendefinition gefragt, d.h. hier muss noch keine Funktionalität implementiert werden.

Aufgabe 2 (1 Punkt). Implementieren Sie einen Konstruktor für die Klasse `SquareMatrix`, der bei Übergabe eines Integers $n > 0$ und eines Initialwertes $v \in \mathbb{R}$ eine $n \times n$ Matrix vom Typ `SquareMatrix` mit Einträgen v erzeugt.

Aufgabe 3 (1 Punkt). Implementieren Sie eine Methode `getN`, die die Dimension $n > 0$ einer $(n \times n)$ Matrix A vom Typ `SquareMatrix` zurückgibt.

Aufgabe 4 (3 Punkte). Implementieren Sie die Methode `setEntry` der Klasse `SquareMatrix`. Denken Sie daran, etwaige Sicherheitsabfragen zu verwenden. Was müssen Sie hierbei beachten, damit Sie die Matrix-Vektor-Multiplikation von `SquareMatrix` auch bei der abgeleiteten Klasse `UpperTriangular` funktioniert?

Aufgabe 5 (4 Punkte). Implementieren Sie eine Matrix-Vektor-Multiplikation, der Sie einen Vektor vom Typ `MyVector` übergeben. Der Rückgabewert der Methode soll ebenfalls vom Typ `MyVector` sein. Zu einer Matrix $A \in \mathbb{R}^{n \times n}$ vom Typ `SquareMatrix` und Vektoren x, b vom Typ `MyVector` soll der Aufruf $x = A * b$ das Ergebnis der Matrix-Vektor-Multiplikation in x speichern.
Hinweis: Die Klasse `MyVector` müssen Sie hierzu nicht extra implementieren. Sie können stattdessen alle im obigen UML-Diagramm dargestellten Methoden einfach verwenden. Evtl. ist die folgende Formel zur Matrix-Vektor-Multiplikation hilfreich. Es gilt

$$b_j = \sum_{k=0}^{n-1} A_{jk} x_k \quad \text{für alle } j = 0, \ldots, n-1.$$

Aufgabe 6 (5 Punkte). Was tut die folgende Funktion `func1` bei Übergabe des Vektors

$$\text{vec} = \begin{pmatrix} 3 \\ 7 \\ 4 \end{pmatrix}$$

und des Integers $j = 2$? Geben Sie tabellarisch wieder, welchen Wert die Variablen zum angegeben Zeitpunkt haben. Erweitern Sie hierfür die untenstehende Tabelle geeignet. Welche Funktionalität wird durch `func1` generell bereitgesellt? Welche Aufgabe erfüllt die Funktion `func2`?

```
double func2(double a){
    if (a<0){
        return -a;
    }else{
        return a;
    }
}

double func1(MyVector &vec, int j){
    double res = 0;
    double val = 0;
    for (int i = 0; i < vec.getLength(); i++){
        if (j == 1){
            val = func2(vec.getEntry(i));
            res = res + val;
        }
        if (j == 2){
            val = vec.getEntry(i)*vec.getEntry(i);
            res = res + val;
        }
        /* Wert der Varialen zu diesem Zeitpunkt */
    }
    return res;
}
```

j	i	val	res

Aufgabe 7 (2 Punkte). Große Objekte wie Vektoren oder Matrizen sollte man generell nicht als Kopie übergeben. Welche zwei Möglichkeiten kennen Sie, um dieses Problem zu lösen? Worin unterscheiden sich diese Optionen?

Aufgabe 8 (2 Punkte). Schreiben Sie die Klassendefinition einer Klasse `UpperTriangular` zur Speicherung von oberen Dreiecksmatrizen, die Sie öffentlich von `SquareMatrix` ableiten. Die Klasse soll speichereffizient arbeiten, d.h. überflüssige Nullen sollen **nicht** gespeichert werden. Eine 3×3 Matrix

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a_{11} & a_{12} \\ 0 & 0 & a_{22} \end{pmatrix}$$

werde also beispielsweise in einem Vektor `entries` mit

$$\texttt{entries} = (a_{00}, a_{01}, a_{11}, a_{02}, a_{12}, a_{22})$$

gespeichert. Die Klasse soll zusätzlich über die folgenden Methoden verfügen:

- überarbeitete Konstruktoren (analog zur Klasse `SquareMatrix`) zum Anlegen einer effizient gespeicherten oberen Dreiecksmatrix
- überarbeitete Methoden `getEntry` und `setEntry` zum Zugriff auf den Speichereffizienten Koeffizientenvektor
- eine Methode `solve` zum Lösen eines oberen Dreieckssystems

Füllen Sie außerdem das obenstehende UML Diagramm analog zu `SquareMatrix` aus.
Hinweis: Analog zu Aufgabe 1 ist hier noch keine Funktionalität zu implementieren.

Aufgabe 9 (2 Punkte). Wie Sie aus der Vorlesung und zahlreichen Übungen wissen, wird der Eintrag A_{jk} einer oberen Dreiecksmatrix bei spaltenweiser Speicherung im Vektor `entries` an der Stelle $\frac{k(k+1)}{2} + j$ gespeichert. Erklären Sie diese Formel und implementieren Sie die Methode `getEntry` der Klasse `UpperTriangular`. Achten Sie auch hier auf etwaige Sicherheitsabfragen.

Aufgabe 10 (6 Punkte). Implementieren Sie die Methode `solve` der Klasse `UpperTriangular` die einen Vektor $b \in \mathbb{R}^n$ vom Typ `MyVector` übernimmt. Die Methode soll nun die Lösung $x \in \mathbb{R}^n$ des oberen Dreieckssystems $A * x = b$ bestimmen und zurückliefern. Prüfen Sie bei etwaigen Divisionen, ob diese auch möglich sind.
Hinweis: Um den Algorithmus herzuleiten, löse man die Matrix-Vektor-Multiplikation mit einer oberen Dreiecksmatrix (siehe obige Formel) nach den x_j auf.